Timex Sinclair User Group NEWSLETTER

JULY/AUG 1988

Ottawa Chapter

EDITOR - M. Dove Volume V Issue I

TABLE OF CONTENTS

Meeting/Ad Rates 0001

Surface Mount Components. 0010

Light Computers.......... 0100

Adapting Programs (Disc). 0110

Pascal Disc Handler...... 0111

Interesting Programs
(Reprints).........10010

ADVERTISING RATES

Members - 50 words or less.
(No charge)

Non-Members - 50 words or less
($5.00)

Commercial

Full page $30.00

Half page $22.00

Quarter page .. $12.00

Eighth page ... $ 6.00

MEETINGS

LOCATION: All meetings are held at the NRC bldg,. 100 Sussex Dr. Ottawa. Check with security when signing in for the room number.

TIME:Meetings are held at 8.00 P.M. the second Tuesday of every month (no meetings in July or August).

FORMAT:
1)-President's Address and introduction of new members.
2)-Software review (max of 20 minutes).
3)-Hardware review (max of 20 minutes).
4)-Adjourment of the formal part of the evening.
5)-Informal: Members discuss common problems, swap programs, and exchange info (tips) etc.

MEETING DATES

APRIL	12th
MAY	10th
JUNE	14th
SEPTEMBER	13th
OCTOBER	11th
NOVEMBER	8th
DECEMBER	13th

MAILING ADDRESS

Please Address all correspondence to:

Michael Dove
The Editor
OHTSUG
2161 East Acres Rd.,
Gloucester, Ontario
Canada K1J 9A6

The following three articles come reprints of articles from Bill Harmer. Thanks Bill for the contributions.

SURFACE MOUNT COMPONENTS

The printed circuit boards that we have been familiar with so far (like the ZX-81/TS1000 and TS2068 mainboards) have been the conventional kind in which integrated circuits (or their sockets) transistors and resistors etc. are mounted with their leads through holes in the board to solder pads or eyelets etched on the other side of the board. This is the way things used to be done before the need to pack more components into smaller spaces (as in portable video cameras etc.) led to a system that used components significantly smaller than those of the older technology. The case that houses a small transistor these days for surface mount technology is hardly bigger than the chip itself. So surface mount technology used increasingly in new microcomputer designs also offers the hardware hacker using low cost or surplus computers that he would not be afraid to hack the opportunity to add his own circuit boards to drive accessories etc. as small as one inch by one inch so that they are easily mounted on the board over a dip socket or using silicon glue right on top of a spare area or cool running component on the board. Printed circuit boards also need not necessarily be drilled, allowing less opportunity to damage them while drilling or missing pads with over-size drills, a problem with the through the board hole mount technology.

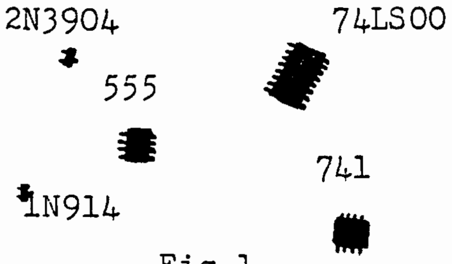

Fig.1
Actual Size S.O. Parts

Looking at Fig. 1 with the surface mount components drawn to scale shows you how they compare to d.i.p. package and TO-92 components that they replace. Similar economies can be realized with the passive components, resistors, capacitors (chip) and aluminum and tantalum electrolytic capacitors are all available in this format.

Table 1 indicates the standard parts that have SO-23, surface mount component equivalents available through a parts supplier like Active Components Sales (with branches in many cities in the U.S. and Canada).

Suppliers of passive components are harder to find. Two of them that advertised recently are: Value Computer, Box 1151, Alta Loma, CA 91701, USA, (714) 987-2414 and Garrett Inst. & Comp./IEU Inc. 3130 Skyway Dr. #104, Santa Maria, CA 93455, USA (805)922-0594. However you need not buy these often expensive passive components to experiment with. Standard 1/8 watt resistors (Radio Shack #271-311) can be used and small capacitors (military square chip for example) and tantalum electrolytic teardrop kind can be adapted for surface mount applications.

2N3904	(NPN)	switching-digital
2N3906	(PNP)	" "
2N2222A	(NPN)	amplifying-linear
2N2907	(PNP)	" "
2N918	(NPN)	R.F. amplifier
1N4148	diode	switching-digital
74LSxx	series	TTL
14000	series	(4000 CMOS)
741	op amp.	
555	timer-pulser	(std.-bipolar)
555 CMOS	timer-pulser	
324	quad op amp.	

Table 1 -Parts Available

How to make a surface mount printed circuit board? The standard ways using an etch pen or Ceresist rub-on lines and pads or drafting tape will work just as in regular p.c.b. work. However since the lines can be finer one must be more careful about flaws and breaks in lines. Also it is wise to use large pads for certain components that will take a little heat. Remember the power rating formula for resistors etc. P=voltage drop x current x 1.4 (safety factor) should be below 1/8 watt for these designs as opposed to the ¼ or ½ watt of the older projects using larger components. Simply reducing current flow in your designs should accomplish this. So p.c.b. making is similar to ordinary p.c.b. making less the hassle (in most cases) of drilling holes though them with fine and fragile drill bits.

Soldering is more delicate and should use a smaller soldering iron than the standard 30 watt pencil. A 15 watt pencil iron like Radio/Shack #64-2051 should be considered or at the very least soldering using an extension tip made my wrapping some solid copper hook-up wire around the tip of an ordinary iron and running it out to ½ inch soldering point away from the iron tip end. And you must be extra careful about soldering adjacent pins in the integrated circuits together and exposing any component to much heat for very long. To hold the component in place while soldering it, you can use masking tape along one side (the opposite to that being soldered of course) or a dab of say silicone glue (but make sure the pins are not up-in-the-air, away from the solder pads).

In repairing equipment with surface mount components you should be aware of some special cautions. For one, you should assume that all components are static sensitive (damagable by stray static from walking accross a carpet for example) unless proven otherwise. Use static-free chemical sprays only. The surface mount components are also much more sensitive to shock (physical shock) and heat damage as are their attachment points to the board. The long leads of conventional components isolate them from physical vibration and shock as well as heat variations much more than the short terminals of surface mount components. So use of heat-guns and freeze spray must be very moderate and careful. And of course static inhibiting freeze spray should be used. Also when removing a component or touching the board with a tool or pulling a board out of a connector mounting one must be careful not to flex the board as this can cause the surface mount components to become detatched or traces or solder joints to break open in a way that the hardier conventional component boards are not so sensitive to, being better anchored with their through-the-board leads and larger, wider copper traces.

Finally, in testing surface mount components, the same tests that can be used on regular boards are still used but when placing the test equipment or multi-meter leads on the leads of the surface mount components you are dealing with two additional problems. The leads are so fine and the terminals so close together that it is much easier to damage or short leads if your hand slips or is sloppy in not getting the probe right on the right component terminal. The other problem is that the component particularly for resistors and capacitors has its guts exposed so to speak so that placing the probe on the end or slipping it accross the end of a component such as a resistor can damage the joint between the leadless terminal and the pill of resistive material or even damage the resitive material changing the resistor's value. The proper place to place the probe on capacitors or resistors with only a metal end rather than a lead is on the pad of the circuit that it is soldered to, as far from the actual solder of the connection as possible to avoid breaking that joint too. So the overall guiding thought in dealing with this new technology is its increased fragility and fineness.

The integrated circuit pins when seen from the top go around the outside starting from pin 1, in counter-clockwise direction. Pin one is often shown by a little indented dot opposite it. In the large, square packages, of LSI/VLSI chips, the corner where pin 1 is found is cut off at an angle not a pointy square like the other corners. For transistors Fig. 2 shows the terminals.

Now for a brief list of part numbers:(small x=any no.) 2N2222A =MMST2222A, 2N3904=MMST3904, 2N3906 = MMST3906, 2N2907= MMST2907A, pow. driver 2N2222A =RXT2222A, 1N4148=RLS4148, 1N4001=RLR4001, Zener diode, 5.1 volt =RLZ5231B, 741 op amp = BA14741AF, dual op amp (LM358 eqiv.)=BA4558F, 74HCxx = BU74HCxxF, 74LSxx =SN74LSxxD, CA400x= BU400xBF, quad op amp (LM324 equiv.)=BA10324F NE555SOIC=(bipolar NE555 timer-pulser).

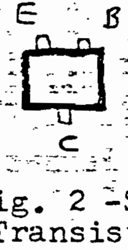

Fig. 2 -SO-23
Transistor
From the Top (Surface)

LIGHT COMPUTERS

by Bill Harmer

When one starts out mentioning light computers, most people are not sure what they are and how they differ from electronic computers. One might explain firstly that it is not the task that makes them different, they would be designed to do the same sort of computing and data processing jobs that electronic computers do currently, but it is how they do it that makes them different. In short, they use light rather than electrons to make them work.

In an electronic computer it is the switching on and off of the flow of electrons that is used to represent the bits and bytes of data and logical relations in computer operations. Thus the key element of a digital computer is the transistor switch. It may be a single transistor in a case as with the older computers of the 1950's and 1960's or it may be that the transistor is one of many on a silicon chip called an integrated circuit. From these switches, Boolean logic operations, routing of data, storage of data in flip-flops or memory cells of MOSFET transistors and arithmetic operations are built up.

A light computer would similarly use switches as its basic building block but the switches would switch light rather than electrons. The light would flow along fibre optic cables of glass rather than metal wires and through light pipes engraved in integrated circuit chips and boards. The actual switching might switch a light beam on or off or switch its frequencies or its polarization to give three possible examples. But the principle of switching light would be the same as with switching electrons in a transistor, one signal controlling the switching of another, typically a small input signal switching a larger output signal.

Are we likely to see a light (or photonic as some say) computer in the near future? The writer wouldn't think that a purely light based computer will be seen for more than a decade or even decades into the future. Why? So far the building of light switching components and photonic (light) integrated circuits has only been done in the laboratory and at too low a level of complexity (only a few gates per integrated circuit) to compete with the current electronic technology by which thousands of gates can be put on a single silicon chip integrated circuit.

What would be the advantage of a light based computer? The chief one is speed. Light simply can be switched faster than electrons and there is nothing like the stray capacitance of electrical circuits to slow it down. In addition possible parallelism (switching of more than one light signal at a time) and immunity to RFI/EMI interference are also mentioned.

What we are likely to see a lot sooner is some form of hybrid light/electronic computer as more fibre optic cables could be made use of in electronic computers. If fibre optic cables were used to wire up the different printed circuit boards of a super-computer, the faster travel of light down the optic cables would speed up the computer by allowing the data to travel more quickly between boards. In supercomputers the time taken to travel down a wire is a considerable limiting factor in data processing speed and for this reason many of the supercomputers are designed with their components arrayed in a circle as close to the physical centre of the computer as possible to cut down on the time wasted in bits and bytes travelling down long wires. But there is a limit to how much this distance can thus be reduced and still make the computer complicated enough to do its job which requires as many boards as possible.

With home computers, light pipes instead of wires would limit RFI (radio frequency interference) problems, preventing our computers from interfering with the use of others home entertainment electronics (like radios) and eliminating the crashes in our own computers, particularly portable models, caused by auto ignition systems and citizens' band radio transmitters.

In addition certain wiring problems for VLSI and ULSI (very dense) integrated circuits might be able to be handled by using tiny fibre optic cables rather than wires to bring signals to and from the surface of the chip. This would not be possible with wires, which can not be made fine enough and must be bonded to rather large (relative to the rest of the components' sizes) pads on the chip.

What use of light components will be see in the near future? The closest to application in the personal computer area is the CD ROM. At the moment compact disks (CD's) are used for music only and the connections on the back of your CD player are meant to be hooked up only to your stereo amplifier. In the future additional connectors could hook up to your t.v. set to play rock videos from the disks and possibly through an RS-232 connector to your personal computer to access digital data on your screen and computer printer. This would be a CD ROM, you can't write on it but you can read from it into your computer. There has been talk of putting the whole Encyclopedia Britannica on a CD disk as well as the phone directory for the whole of Canada, or a shelf of medica or technical jornals. These could be accessed on your computer screen.

A laser RAM style of system, really a laser hard disk has been developed by Hitachi for industrial use. It uses a laser to read the magneticcally encoded information on the disk. You can both read to and write on the disk with your computer. A cheaper version may come along for us.

Fibre optics are to be used to hook up microcomputers in offices into a LAN, or local area network, so that they can share the use of periphera and exchange messages and data. Fibre optic cables are less cumbersome in this use and do not have the RFI problems of metal cables.

In looking at other attempts to speed up computer operation, two of them would seem to dove-tail. The development of gallium arsenide (GaAs) electronic integrated circuit chips and the fact that since gallium arsenide is transparent it makes the ideal semiconductor to synthesize both electronic and optical devices (like LED's and photodiodes and transistors on the same chip. It is a natural for hybrid light/electronic devices and computers and its technology although proving harder to use than silicon chip methods is bound to show progress with the number of labs, including our local Northern Telcom/Bell/Northern, working on perfecting it.

So now, in addition to the present technology of electronic computers using silicon chip integrated circuits, we can see on the horizon the possibilility of computers using superconductivity and Josephson junction technology as well as light rather than electron based data representation. At the moment it is any one's guess which will be the prevailing technology in the computers of the future.

Progress: The first digital computers used telephone swtches to make them work. Now telephone switches use computers to make them work.

ADAPTING PROGRAMS FOR DISK DRIVE

by Bill Harmer

When the writer got his disk drive from Larken Electronics, the task at hand was to transfer as many as possible of the programs to disk. A typical cassette autosave routine like:

```
9000 REM CASSETTE SAVE
9010 CLS
9020 PRINT"CASSETTE SAVE BEGINS"
9030 PAUSE  200
9040 CLS
9050 SAVE"PROGRAM ONE"
9060 CLS
9070 GOTO 1
```

... was left in and a disk save routine was added like:

```
9500 REM DISK SAVE
9510 CLS
9520 PRINT USR 14336
9530 REM SAVE "PROG1.B1"
9540 SLOW
9550 CLS
9560 GOTO 1
```

Then with these two routines within the program but off by themselves simply keying in the immediate command GOTO 9000 for a cassette save and GOTO 9500 for a disk save, the appropriate save routine could be accessed so that on loading from cassette or disk the next time the program would be self-starting at line 1.

Further program refinements allowed on some programs the access of a GOTO 9000 or GOTO 9500 by choosing on a menu the option of saving on cassette or disk. So with a few simple additions the programs could be adapted to run conveniently on the Larken disk system and still allow saving on cassette for swapping programs or back-up.

Another disk save routine has been used to double check the save on the diskette by allowing the option of doing a disk directory with the press of a single key once the GOTO 9500 routine is invoked. Sometimes the disk operating system will appear to save your program but not really do it and it indicates this in the directory by showing a 0 opposite the file label to indicate that no sectors have been assigned to this file and that in fact it hasn't been saved. The routine is as follows:

```
9500 REM DISK SAVE
9510 CLS
9520 PRINT USR 14336
9530 REM SAVE "PROG1.B1"
9540 SLOW
9550 PRINT "VERIFY? (Y OR N)"
9560 IF INKEY$<>"Y" AND INKEY$<>"" THEN GOTO 9600
9570 IF INKEY$="" THEN GOTO 9560
9580 PRINT USR 14336
9590 REM DIRE
9600 CLS
9610 GOTO 1
```

Sometimes there is a false "yes" response with the INKEY$ but generally this routine saves a lot of headaches and requires only a single keystroke to use. An alternative routine uses INPUT instead of INKEY$. It is amazing to think that our little computer allows self-starting programs that even the Commodore 64 doesn't allow normally.

PASCAL DISC HANDLER

David has supplied us with many interesting articles in the past. Larry Kenny is of course the owner and distributer of the Larken disc drive.

PASCAL DISK HANDLER
FOR THE
LARKEN DISK OPRERATING SYSTEM

ARTICLE BY
DAVID SOLLY

PROGRAMS BY
DAVID SOLLY
AND
LARRY KENNY

Access to the Larken Disk Drive System from within a compiled HiSoft(TM) Pascal program is now possible thanks to the procedures developed by David Solly and Larry Kenny. This article demonstrates how to install these procedures and gives a practical demonstration on how they may be used within a simple directory program. The procedures described in this article are valid for both the Timex Sinclair 2068 and the ZX Spectrum versions of HiSoft Pascal.

I have owned HiSoft Pascal, (henceforth Pascal), for the Timex Sinclair 2068 and the ZX Spectrum for a number of years now but the drawback with Pascal, as it is implemented on Sinclair computers, has been that there was no way to access a DOS from within a compiled program. It seemed ridiculous to put the effort into creating programs which ran like machine code once the source code was debugged, compiled and transfered to disk only to be forced to go back to a plodding tape operating system, (henceforth TOS), when I needed to save or enter data from within the compiled program. Finally I sat down and did some serious study of the problem, some head scratching, some question asking (especially of Larry Kenny, who is the creator of the Larken DOS, and Ken Schieman) and some experimentation. The result of this is the creation of the Pascal disk handler described below.

The needs of LKDOS are the same as for the TOS. You are required to provide the name of your program, the starting address of the program and the lenght of the program. This is exactly what happens when you type within Basic:

```
SAVE "Myprogram" CODE start, length.
```

Pascal also passes the same information to the TOS when it saves out a variable through the use of the procedure:

```
TOUT ('Myprogram', ADDR (the name of the variable being
saved), SIZE (the name of the variable being saved));
```

The crux of the problem is that, although both languages generate identical information, LKDOS can extract it only from within the Basic environment. Therefore, additional programming is need in order to transfer the save or load parameters to LKDOS from within the Pascal environment.

The first steps towards transfering the required information form Pascal to LKDOS are to store the name, the start address, and the length of the varialble to be saved or loaded in a safe area of the RAM. This is a simple matter thanks to the very versatile POKE() function in Pascal. Unlike the POKE command available in Basic, this Pascal function can DOKE a number or POKE the contents of a whole character array starting at a given address. The first fifteen bytes of the printer buffer proved to be the safest area to which all the transfers can be achieved with the following lines:

```
POKE (23300, 'Myprogram');
```

```
POKE (23311, ADDR (name of variable));
```

and

```
POKE (23313, SIZE (name of variable));
```

You can see these lines in a modified form in the demonstration program within the procedures GETNAME, DISKSAVE and DISKLOAD.

Upon completion of these steps the next step is to copy the information from its storage place in the RAM to the appropriate routines in the LKDOS. Again to HiSoft includes within their implementation of Pascal the procedure INLINE() wich allows us to embed Z80 machine code within a Pascal procedure to accomplish our ends.

The following is a disassembly of the machine code used in the procedure DOUT

```
00010 NAME      EQU 23300
00020 PROGNM    EQU 8226
00030 TEMP4     EQU 8243
00040 TEMP2     EQU 8241
00050 NMIF      EQU 8194
00060 ADDR      EQU 23311
00070 SIZE      EQU 23313
00080 SV1       EQU 0204
00090 SV2       EQU 0207
00100           ORG 40000
00110           DI                  ; Disable interrupt
00120           CALL 98             ; Turn on LKDOS cartridge
00130           LD HL,NAME          ; Transfer file name
00140           LD DE,PROGNM        ; to prognm
00150           LD BC,09
00160 MOVE      LD A,(HL)           ; Loop to catch any
00170           CP 0                ; occurance of CHR$ 0
00180           JR NZ,NZERO
00190           LD (HL),32          ; & replace with a space
00200 NZERO     LDI
00210           LD A,B
00220           OR C
00230           JR NZ,MOVE
00240           LD A,11
00250           LD (NMIF),A
00260           CALL SV1            ; Save name
00270           LD HL,(ADDR)        ; Retrieve start address
00280           LD (TEMP4),HL       ; Set start address
00290           LD HL,(SIZE)        ; Retrieve length of save
00300           LD (TEMP2),HL       ; Set length of save
00310           CALL SV2            ; Second save data
00320           LD A,(100)          ; Exit cartridge
00330           EI                  ; Enable interrupt
```

(Program provided by Larry Kenny of Larken Electronics, #2
Navan, Ontario, Canada K4B 1H9, Tel: (613)-835-2680)
(Listing made using Zeus Assembler)

The procedure DIN contains the same code except that SV1 and SV2 are replaced with LD1 EQU 198 and LD2 EQU 201. The MOVE loop insures that the file name is padded out with the correct number of spaces so that it totals nine characters and spaces otherwise the file name may be corrupted and fail to reload.

The following listing demonstrates how all the procedures described above are used within a complete Pascal program. The object of the program is to create a simple telephone directory which will allow you to store ten names and numbers, read the information stored in the directory, and read and write the information stored in the directory to disk using the Larken LKDOS. There is also a summation of this article contained in the procedure SONGANDDANCE.

(Pascal source code listing begins overleaf)

```
 10 {PROGRAM BY:               }
 20 {DAVID SOLLY               }
 30 {1402-1545 ALTA VISTA DRIVE}
 40 {OTTAWA, ONTARIO           }
 50 {CANADA K1G 3P4            }
 60 {TEL: (613)-731-2120       }
 70
 80
 90 {THIS PORGRAM IS FOR       }
100 {DEMONSTRATING HOW THE     }
110 {LARKEN DISK DRIVE MAY BE  }
120 {ACCESSED FOR STORING AND  }
130 {RETRIEVING DATA WITHIN A  }
140 {HISOFT(TM) PASCAL PROGRAM.}
150 {THE PROCEDURES DESCRIBED  }
160 {IN THIS PROGRAM ARE VALID }
170 {FOR BOTH THE ZX SPECTRUM  }
180 {AND  THE TIMEX SINCLAIR   }
190 {2068 VERSIONS OF HISOFT   }
200 {PASCL.                    }
210
220
230 PROGRAM LARKENDEMO;
240
250 CONST
260   LENGTH = 10;
270   PN = 23300; {ADDRESS WHERE THE NAME FOR THE SAVE/LOAD}
280               {ROUTINES IS STORED}
290   PB = 23311; {CONTAINS THE ADDRESS WHERE THE DATA BEGINS}
300   PS = 23313; {CONTAINS THE NUMBER OF BYTES USED BY THE DATA}
310   CAPSLOCK = 23658; {ADDRESS OF THE CAP SHIFT LOCK CONTROL}
320   ON  = 8; {SWITCHES CAPSLOCK ON}
330   OFF = 0; {SWITCHES CAPSLOCK OFF}
340
350
360 TYPE
370   ENTRY = RECORD
380             NAME : ARRAY [1..10] OF CHAR;
390             NUMBER : ARRAY [1..10] OF CHAR;
400           END;
410
420
430 VAR
440   DIRECTORY : ARRAY [1..LENGTH] OF ENTRY;
450   I : INTEGER;
460   FINISHED : BOOLEAN;
470   ANS : CHAR;
480
490
500
```

```
 510
 520 PROCEDURE DOUT; {INVOKES LKDOS WRITE}
 530   BEGIN
 540     INLINE  (#F3, #CD, #62, #00, #21, #04, #5B, #11,
 550              #22, #20, #01, #09, #00, #7E, #FE, #00,
 560              #20, #02, #36, #20, #ED, #A0, #78, #B1,
 570              #20, #F3, #3E, #0B, #32, #02, #20, #CD,
 580              #CC, #00, #2A, #0F, #5B, #22, #33, #20,
 590              #2A, #11, #5B, #22, #31, #20, #CD, #CF,
 600              #00, #3A, #64, #00, #FB);
 610 END;
 620
 630
 640 PROCEDURE DIN; {INVOKES LKDOS READ}
 650   BEGIN
 660     INLINE  (#F3, #CD, #62, #00, #21, #04, #5B, #11,
 670              #22, #20, #01, #09, #00, #7E, #FE, #00,
 680              #20, #02, #36, #20, #ED, #A0, #78, #B1,
 690              #20, #F3, #3E, #0B, #32, #02, #20, #CD,
 700              #C6, #00, #2A, #0F, #5B, #22, #33, #20,
 710              #2A, #11, #5B, #22, #31, #20, #CD, #C9,
 720              #00, #3A, #64, #00, #FB);
 730 END;
 740
 750
 760 PROCEDURE GETNAME;
 770 VAR
 780   PROGNM : ARRAY [1..9] OF CHAR;
 790
 800   BEGIN
 810     WRITELN;
 820     POKE (CAPSLOCK, OFF);
 830     WRITELN ('NAME FOR DISK OPERATION? ');
 840     WRITELN;
 850     WRITE ('9 CHARACTERS MAXIMUM:  ');
 860     READLN;
 870     READLN (PROGNM);
 880     POKE (PN, PROGNM); {STORES THE NAME IN RAM}
 890   END;
 900
 910
 920 PROCEDURE DISKSAVE;
 930 BEGIN
 940   PAGE;
 950   WRITELN;
 960   WRITELN ('LARKEN DISK SAVE ROUTINE');
 970   WRITELN;
 980   GETNAME;
 990   POKE (PB, ADDR (DIRECTORY)); {START ADDRESS FROM WHERE THE}
1000                                {DATA IS TO BE SAVED}
1010   POKE (PS, SIZE (DIRECTORY)); {NUMBER OF DATA BYTES TO SAVE}
1020   DOUT;
1030   WRITELN ('SAVED!');
1040   FOR I := 1 TO 100 DO {PAUSE LOOP}
1050 END;
1060
```

```

PROCEDURE DISKLOAD;
BEGIN
  PAGE;
  WRITELN;
  WRITELN ('LARKEN DISK LOAD ROUTINE');
  WRITELN;
  GETNAME;
  POKE (PB, ADDR (DIRECTORY)); {START ADDRESS AT WHICH THE}
                                {DATA IS TO BE LOADED}
  POKE (PS, SIZE (DIRECTORY)); {NUMBER OF DATA BYTES TO LOAD}
  DIN;
  WRITELN ('LOADED!');
  FOR I := 1 TO 100 DO {PAUSE LOOP}
END;


PROCEDURE FILLDIRECTORY;
BEGIN
  PAGE;
  POKE (CAPSLOCK, OFF);
  FOR I := 1 TO LENGTH DO
    BEGIN
      WITH DIRECTORY [I] DO
        BEGIN
          WRITELN ('ENTRY NO. ', I, ' OF ', LENGTH);
          WRITELN;
          WRITE ('NAME PLEASE (10 CHARS) ');
          READLN;
          READ (NAME);
          WRITE ('NUMBER PLEASE (10 CHARS) ');
          READLN;
          READ (NUMBER)
        END
     END;
    WRITELN ('DIRECTORY FULL ');
    WRITELN;
    WRITELN ('HIT ANY KEY TO CONTINUE');
    READLN
END;

```

```
1480
1490 PROCEDURE READDIRECTORY;
1500 BEGIN
1510   PAGE;
1520   POKE (CAPSLOCK, OFF);
1530   FOR I := 1 TO LENGTH DO
1540     BEGIN
1550       WITH DIRECTORY [I] DO
1560         BEGIN
1570           WRITELN (NAME,'     ', NUMBER);
1580           WRITELN;
1590         END
1600     END;
1610    WRITELN ('END OF DIRECTORY');
1620    WRITELN ('HIT ANY KEY TO CONTINUE');
1630    READLN
1640 END;
1650
1660
1670 PROCEDURE SONGANDDANCE;
1680   BEGIN
1690     PAGE;
1700     WRITELN ('Larken Disk Access Routine');
1710     WRITELN ('for');
1720     WRITELN ('HiSoft(TM) Pascal');
1730     WRITELN ('for the');
1740     WRITELN ('ZX Spectrum');
1750     WRITELN ('and the');
1760     WRITELN ('Timex Sinclair 2068');
1770     WRITELN;
1780     WRITELN ('by');
1790     WRITELN ('David Solly');
1800     WRITELN ('and');
1810     WRITELN ('Larry Kenny');
1820     WRITELN;
1830     WRITELN ('Hit any key to continue');
1840     READLN;
1850     PAGE;
1860     WRITELN ('Many ZX Spectrum and TS 2068');
1870     WRITELN ('programers have long wanted to');
1880     WRITELN ('do serious programing in other');
1890     WRITELN ('languages than the resident');
1900     WRITELN ('Sinclair Basic. Although such');
1910     WRITELN ('languages as Forth, Logo, C,');
1920     WRITELN ('Prolog and Pascal have long been');
1930     WRITELN ('available to Sinclair users one');
1940     WRITELN ('of the main drawbacks for');
1950     WRITELN ('serious programing in these');
1960     WRITELN ('languages has been the lack of');
1970     WRITELN ('disk I/O routines. This program');
1980     WRITELN ('will demonstrate how the Larken');
1990     WRITELN ('disk drive system may be');
2000     WRITELN ('accessed for storing and ');
2010     WRITELN ('retrieving data within a');
2020     WRITELN ('HiSoft(TM) Pascal program. The');
```

```
2030     WRITELN ('procedures described in this ');
2040     WRITELN;
2050     WRITELN ('Hit any key to continue');
2060     READLN;
2070     PAGE;
2080     WRITELN ('program are valid for both the');
2090     WRITELN ('ZX Spectrum and the Timex');
2100     WRITELN ('Sinclair 2068');
2110     WRITELN;
2120     WRITELN ('The two disk drive procedures');
2130     WRITELN ('are found in the procedures DIN,');
2140     WRITELN ('and DOUT. A third procedure,');
2150     WRITELN ('called GETNAME, supplies the');
2160     WRITELN ('above procedures with a name for');
2170     WRITELN ('storing or retrieving from the');
2180     WRITELN ('disk. All these procedures work');
2190     WRITELN ('in conjunction with the resident');
2200     WRITELN ('procedures ADDR(), SIZE(), and');
2210     WRITELN ('POKE()');
2220     WRITELN;
2230     WRITELN ('The authors hope that these');
2240     WRITELN ('procedures will stimulate Pascal');
2250     WRITELN ('programing for the ZX Spectrum');
2260     WRITELN;
2270     WRITELN ('Hit any key to continue');
2280     READLN;
2290     PAGE;
2300     WRITELN ('and the Timex Sinclair 2068 and');
2310     WRITELN ('encourage other Sinclairests to');
2320     WRITELN ('write disk routines for the');
2330     WRITELN ('other languages mentioned');
2340     WRITELN ('above.');
2350     WRITELN;
2360     WRITELN ('David Solly');
2370     WRITELN ('Larry Kenny');
2380     WRITELN;
2390     WRITELN;
2400     WRITELN ('Hit any key to continue');
2410     READLN
2420   END;
2430
2440
2450 BEGIN {BODY OF THE PROGRAM}
2460   REPEAT
2470     PAGE;
2480     POKE (CAPSLOCK, ON);
2490     FINISHED := FALSE;
2500     WRITELN;
2510     WRITELN;
2520     WRITELN ('MENU');
2530     WRITELN;
2540     WRITELN ('SELECT ONE OF THE FOLLOWING');
2550     WRITELN ('OPTIONS ');
2560     WRITELN;
```

```
    WRITELN;
    WRITELN ('1) READ THE INTRODUCTION');
    WRITELN ('2) CREATE DATA');
    WRITELN ('3) READ DATA');
    WRITELN ('4) SAVE DATE TO DISK');
    WRITELN ('5) LOAD DATA FROM DISK');
    WRITELN ('6) EXIT THE PROGAM');
    WRITELN;
    WRITELN;
    WRITE ('MAKE YOUR SELECTION ');
    READLN;
    READ (ANS);


    CASE ANS OF
     '1' : SONGANDDANCE;
     '2' : FILLDIRECTORY;
     '3' : READDIRECTORY;
     '4' : DISKSAVE;
     '5' : DISKLOAD;
     '6' : FINISHED := TRUE
    END;


  UNTIL FINISHED = TRUE;


    {FINALE}
    PAGE;
    WRITELN;
    WRITELN;
    WRITELN ('END OF DEMONSTRATION');
    WRITELN;
    WRITELN
END.
```

The Pascal disk handler described in this article and program works only within a compiled Pascal program and, regretfull , can not be used to save Pascal source code. The procedures DIN, DOUT, and GETNAME are completely modular. They may be copied directly from this program into the appropriate section of any Pascal program you care to write which requires access to LKDOS. The procedures DISKSAVE and DISKLOAD may also be used but remember to change the name within the parentheses of ADDR() and SIZE() to the name of the variable within your program that you wish to save or load. The ".C" extension required by data file saves within Basic when using LKDOS is not required by the Pascal disk handler, however, it may be good practice to use the extension ".P" to indicate that the data that has been saved is intended for a Pascal program rather than a Basic program. All the LKDOS error codes are operational and will stop your program without crashing providing that the compiled code is accessed through a Basic loader program, that a PRINT USR is used rather than RANDOMIZE USR and that there is at least one line of Basic after the USR call. Example:

```
  10 REM TYPICAL COMPILED PASCAL PROGRAM LOADER
  20 BORDER 0: PAPER 0: INK 7: CLS
  30 PRINT #4: LOAD "MYPROG.C1" CODE 27000
  40 CLS: PRINT USR 27000
  50 STOP
9000 REM SAVE LOADER TO DISK
9010 PRINT #4: SAVE "LOADER.B1" LINE 1
```

On behalf of Larry and myself I hope that this Pascal disk handler will prove to be useful to all who wish to do serious programming in Pascal which requires disk access.

David Solly
Ottawa, Canada,
June 15, 1988

The following articles are reprints of interesting prograams found in other newsmagasines. The first two are from Timelinz (Nov/Dec 1987) while the last program comes from the May/Jun 1988 issue of Syncware News.

TS/1000 VERIFY

by DAVID NOWOTNIK

With only 8K of ROM in the TS/1000 it's little wonder that it hasn't a verify command. This little program will take care of that omission.

This routine is based on the LOAD routine in ROM. The change is when a byte is read off the tape. Instead of putting the byte into the approprite place in RAM, it is compared with the current byte at that address. If their is not a match, then the routine exits with an error message (R/0).

If all bytes match and the verification was successful, with no mis-match, than the 0/0 message will be returned at the end of the routine.

First enter this machine code loading routing, with 135 spaces or characters in the REM statment.

```
  10 REM ......(135 SPACES).....
  20 LET X=16514
  30 INPUT A$
  40 IF A$="S" THEN STOP
  50 LET J=16*(CODE A$-28)+CODE
A$(2)-28
  60 IF PEEK X=27 THEN POKE X,J
  70 LET X=X+1
  80 GO TO 30
```

Next RUN the program and enter these HEX digits.

```
CD 23 0F 37 11 00 00 CB 12 CB
0A CD 10 7C 18 F5 0E 01 06 00
3E 7F DB FE D3 FF 1F 30 49 17
17 38 28 10 F1 F1 BA D2 7A 7C
62 6B CD 10 7C CB 7A 79 20 03
BE 20 D6 23 17 30 F1 FD 34 15
21 09 40 50 CD 10 7C 00 CD 6C
7C 13 F5 D5 1E 94 06 1A 1D DB
FE 17 C8 7B 7B 38 F5 10 F5 D1
20 04 FE 56 30 B2 3F C8 11 30
AD C9 7A A7 28 BB CF 0C EB 21
7C 40 37 ED 52 30 06 1A B9 28
02 CF 1A 13 2A 14 40 37 ED 52
EB D0
```

Now the routine is in the long REM statment, and you can delete lines 20 through 80. DO NOT delete line 10. After these lines are deleted, add the next lines, 20 through 70. Then SAVE this before you go further, it is your VERIFY program.

```
20 LET X=16514
30 FOR I=31744 TO 31878
40 POKE I,PEEK X
50 LET X=X+1
60 NEXT I
70 NEW
```

To use the VERIFY routine, it must be loaded into your TS/1000, above RAM-TOP, before any other program. First lower RAM-TOP with these three direct commands....

```
     POKE 16388,123 -- ENTER
     POKE 16389,255 -- ENTER
          NEW -- ENTER
```

Now you may LOAD and RUN your VERIFY routine.

You now can type in your BASIC program. When you are ready to SAVE it, just SAVE it to tape as you would normally SAVE it. To VARIFY, rewind the tape to the start of the program, and type in, in direct commanded.

```
          RAND USR 31744
```

And press play on your recorder and ENTER on your computer

If you saved the program with variables, then CLEARed them before verifying, or changed the variables in any way then you may get a verify error (R/0). Otherwise, if all is well, you'll get an 0/0 message to tell you that your program has been VERIFIED.

From the Sept. 1987 Plotter N/L
of Clackamas County, Oregon.

A QUICKY

DICK WAGNER

Are you tempted to use UNLESS as a conditional? It would be nice and you can get the same result as: XXX IF NOT (condition) THEN (statement) like this:

```
10 LET C$="Y" UNLESS INKEY $="N"
```

so use this replacement form:

```
10 IF NOT (INKEY $="N") THEN LET
C$="Y"
```

How about the modifier UNTIL? Sinclair BASIC doesn't support UNTIL as in:

```
10 LET Z=Z-21 UNTIL Y<=52
```

try:

```
10 LET Z=Z-21
20 IF NOT (Y<=52) THEN GOTO 10
```

LETTER SCRAMBLE

Basil Loves Joey

Basil Wentworth
TS1000 or TS2068

This is a little game that challenges the player's ability to analyze patterns. The LISTing given is for TS2068; the modifications necessary for the TS1000 are given in the text.

Don't panic. I'm not getting more fickle in my old age. "Joey" is a name that Jocelyn's family inflicted on her when she was very young, and she hasn't ever been able to shake it entirely. And this game needs a four-letter name, as you will see.

Actually, it needs three words--two of five letters each, and one of four letters. So the text could have been

```
  GAMES
  ZX81
  PLAYS
```

It doesn't matter in what order the words occur. In fact, Jocelyn insists on interpreting this game as JOEY LOVES BASIL. I don't mind.

I've seen "hardware" analogs of this game for sale in drug stores. A little matrix holding a number of square tiles, which are free to slide left and right, up and down. And one square vacant, to allow maneuvering room. The trick is to slide them around so that they are in some specified order--whether to make a map of Europe, or an alphabet, or whatever.

When you RUN the program, the TS2068 gives you a courtesy message while the text is being arranged in random order. I haven't been able to get this effect on the TS1000, due to the blanking of screen while it is calculating. But the delay is only about 10 seconds, anyway.

When the randomizing process is complete, the scrambled text is displayed, as shown in the illustration. Pressing UNSHIFTED 5 through 8 makes the asterisk move in the direction of the respective keyboard arrow, changing places with its neighboring text. Unless the asterisk would move "off scale" as a result, in which case a warning message is displayed, and the command is otherwise ignored.

The text to be used is entered as A$ at the beginning of the program, with no spaces, and with an asterisk appearing as the last (15th) character. The program is straightforward from there.

It would have been possible to simplify the program a bit, except that I wanted to keep the programs for the two computers identical, as far as possible. However, anyone wanting to improve on the program is certainly welcome to do so. I'd like your comments.

The changes to be made for the TS1000 are as follows:

Add

```
  5 FAST
175 SLOW
205 FAST
```

Change the 1000 section to read

```
1000 SLOW
1005 FOR F=1 TO 3
1010 FOR G=1 TO 10
1015 NEXT G
1020 PRINT AT 20,0;"  sorry--ill
egal move  " (INVERSE text)
1030 FOR G=1 TO 10
1035 NEXT G
1040 PRINT AT 20,0;"  SORRY--ILL
EGAL MOVE  "
1050 NEXT F
1055 FAST
1060 GOTO 170
```

And the SAVE section:

```
9010 SAVE "BLJ"
9020 RUN
```

The do-nothing loops at lines 1010-1015 and 1030-1035 are delays, to give the error message time to register. And the outer loop, from 1005 to 1050, give more or less the same effect as the FLASH operation on the TS2068.

```
  10 LET A$="BASILLOVESJOEY*"
  20 DIM B$(15)
  30 CLS
  40 PRINT AT 3,5;"BE WITH YOU I
N A MOMENT"
  50 FOR F=1 TO 15
  60 LET N=INT (15*RND+1)
  70 IF B$(N)<>" " THEN GO TO 60
  80 LET B$(N)=A$(F)
 100 NEXT F
 110 CLS
 120 FOR F=1 TO 5
 130 PRINT AT 5,3*F;B$(F)
 140 PRINT AT 8,3*F;B$(F+5)
 150 PRINT AT 11,3*F;B$(F+10)
 160 NEXT F
 170 PRINT AT 20,0;"UNSHIFTED 5-
8 MOVE STAR"
 180 IF INKEY$<>"" THEN GO TO 18
0
 190 IF INKEY$="" THEN GO TO 190
 200 IF INKEY$<>"5" AND INKEY$<>
"6" AND INKEY$<>"7" AND INKEY$<>
"8" THEN GO TO 180
 210 GO TO 100*VAL INKEY$
 500 IF N=1 OR N=6 OR N=11 THEN
GO TO 1000
 510 LET B$(N)=B$(N-1)
 520 LET B$(N-1)="*"
 530 LET N=N-1
 590 GO TO 110
 600 IF N>10 THEN GO TO 1000
 610 LET B$(N)=B$(N+5)
 620 LET B$(N+5)="*"
 630 LET N=N+5
 690 GO TO 110
 700 IF N<6 THEN GO TO 1000
 710 LET B$(N)=B$(N-5)
 720 LET B$(N-5)="*"
 730 LET N=N-5
 790 GO TO 110
 800 IF N=5 OR N=10 OR N=15 THEN
 GO TO 1000
 810 LET B$(N)=B$(N+1)
 820 LET B$(N+1)="*"
 830 LET N=N+1
 890 GO TO 110
1010 PRINT AT 20,0; FLASH 1;"  S
ORRY--ILLEGAL MOVE  "
1015 FOR G=1 TO 300: NEXT G
1060 GO TO 170
8999 STOP
9000 REM SAVE
9010 SAVE "BLJ" LINE 1
9020 BEEP .3,12: BEEP 1,12
9030 CLS : PRINT ''''"   RE-RUN T
APE TO VERIFY "
9040 VERIFY "BLJ"
9050 CLS : PRINT ''''"   VERIFY
OK"
9060 BEEP .3,16: BEEP 1,16
```

TS2068 BASIL LOVES JOEY

TSUG

1987-1988

NAME	ADDRESS	TELEPHONE
STU BIRD	11 SHADY BRANCH TRAIL	836-7611
MICHAEL DOVE	2161 EAST ACRES RD	745-1779
LEO FOSS	1171-2402 AMBLESIDE	820-0010
BILL HARMER	97 RUSKIN ST.	722-7184
BOB HEWENS	2435 THORSON AVE.	828-8802
ROGER HILL	170A CRAIG HENRY DRIVE	723-8822
DAVID HINSON	153/2166 LOYOLA	744-4611
CLEM HOLDEN	83 RENFREW AVE.	235-2139
IRIS T. HOLDER	84 BRADFORD ST.	828-2329
J.W. JARMASZ	767 CLAUDE ST.	745-8159
ORVILLE KENNY	1609-440 GLOUCESTER ST.	232-8460
MORRIE KONICK	302-1081 AMBLESIDE DR.	820-1523
JOHN MATHEWSON	1852 APPLEFORD ST.	746-7869
FRANK MACDONALD	RR#2 WAKEFIELD	459-1162
	907-23 CHARION HULL	819 776-3080
ALAN MOORE	1268 MAITLAND AVE.	226-7118
ISAAC MORLAND	245 IRVING PLACE	725-9208
DALE MORLAND	" "	" "
RODERIC PONTEFRACT		224-7959
COLIN RAYMOND-JONES	27 LAIRD ST.	224-4547
HARRY ROSEWARNE	905 ROSEVIEW AVE.	828-1440
K.D. SCHIEMAN	18 COLUMBUS	749-6277
DAVID SOLLY	1545 ALTA VISTS DR. APT 1402	731-2120